InRule® Deployment Brief: Disconnected Mobile Apps

Use Case

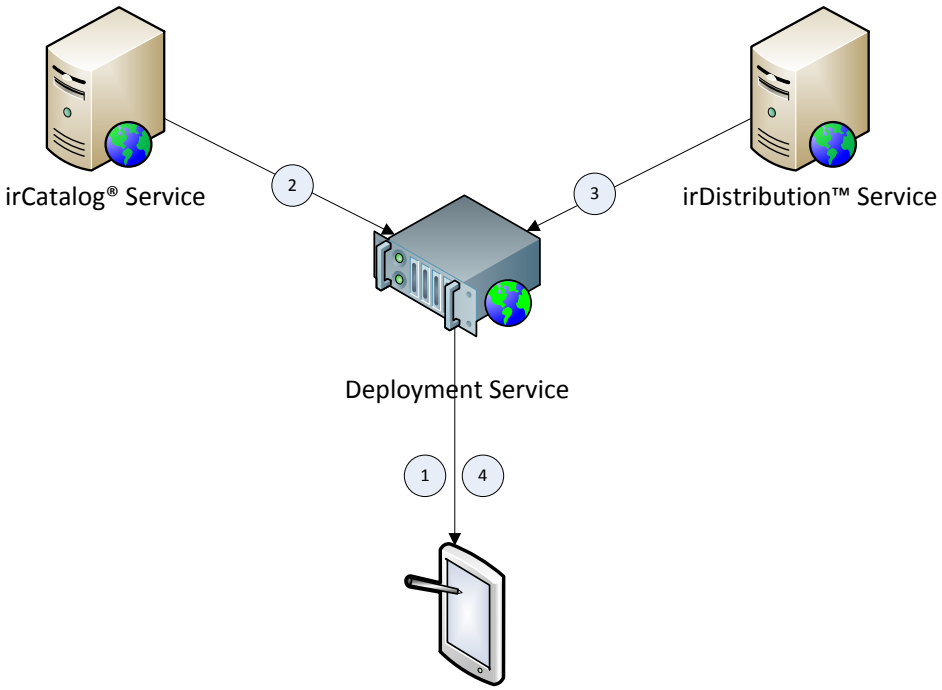
InRule® is often used in applications that support field personnel such as salespeople, service providers and healthcare workers. Increasingly this class of users expects to do their work on tablets instead of PCs. They also bring their own devices to work and expect their mission critical apps to work without network connectivity.

To support these scenarios enterprise architects create hybrid mobile applications that have the ability to run in disconnected mode. Since a hybrid mobile application is simply a web application deployed in a native app container, InRule *for JavaScript* is a viable option for using a business rule management system (BRMS) in multi-platform, disconnected scenarios. When rules are packaged for distribution by the irDistribution™ service the output is a single JS file that contains the rules, rule engine and API for interacting with the engine.

This deployment brief covers how to architect for mobile apps that pull rule updates on demand when connected to a network. See the "For More Information" section at the end of this brief to locate other related deployment and integration briefs.

Getting Updates from a Deployment Service

The following diagram provides a high-level view of a deployment service architecture. We recommend creating a service to handle update requests instead of having the app call the two services directly.



Step 1

The device is running in connected mode and requests a rule update from a custom deployment service you have created. The service can either be accessed via the internet or from an enterprise network and is a good candidate for cloud hosting.

Following is an example of what the call from the app to your deployment service might look like:

```
            function getRules() {
                $("#WaitDialog").removeClass('hidden');
                $.get("http://localhost/javascriptruleservice/api/Rule/HealthInsurance",
function (data) {
                    eval(data.Script);
                    window.inrule = inrule;
                    document.getElementById('ruleButton').click();
                    $("#WaitDialog").addClass('hidden');
                });
            }
```

Step 2

The deployment service retrieves the latest version of the rules from the irCatalog® service as shown in the following C# code. Note that the rules can also be retrieved from a file location. A third option, not shown, is to retrieve the rules as XML from a source control system.

```
        private RuleApplicationDef GetRuleApplication(string ruleAppName, int revision)
        {
            RuleApplicationDef ruleAppDef;
            var useCatalog = true;
            if (useCatalog)
            {
                using (var connection = new RuleCatalogConnection(new
Uri("http://localhost/InRuleCatalogService/service.svc"), new TimeSpan(0, 0, 60),
Settings.Default.CatalogUser, Settings.Default.CatalogPassword))
                {
                    if (revision == 0) // get latest
                    {
                        ruleAppDef = connection.GetLatestRuleAppRevision(ruleAppName);
                    }
                    else
                    {
                        var ruleAppref = connection.GetRuleAppRef(ruleAppName, revision);
                        ruleAppDef =
connection.GetSpecificRuleAppRevision(ruleAppref.Guid, ruleAppref.PublicRevision);
                    }
                }
            }
            else
            {
                var path = AppDomain.CurrentDomain.GetData("DataDirectory") + "\\" +
ruleAppName + ".ruleapp";
                ruleAppDef = RuleApplicationDef.Load(path);
            }

            return ruleAppDef;
        }
```

Step 3

The deployment service then passes the rule application to the cloud-based irDistribution service via a web service call. The following C# code provides an example of how that could be done.

```
        public string GetJavaScriptFromRuleApp(string ruleAppXml, bool turnOnLogging)
        {
            var client = new HttpClient();

            using (var mpfdc = new MultipartFormDataContent())
            {
                byte[] byteArray = System.Text.Encoding.UTF8.GetBytes(ruleAppXml);
                var stream = new MemoryStream(byteArray);
                var httpContent = new StreamContent(stream);
                mpfdc.Add(httpContent, "ruleApplication", "Healthinsurance.ruleapp");
                client.DefaultRequestHeaders.Accept.Add(new
System.Net.Http.Headers.MediaTypeWithQualityHeaderValue("multipart/form-data"));
                string selectedVersion = null;
                var queryString = new System.Text.StringBuilder();


                queryString.AppendFormat("?logOption=None&subscription-key={0}", "[your
key here]");

                var result =
client.PostAsync("https://irjavascriptpackagingproduction.azure-api.net/" +
selectedVersion + "package" + queryString, mpfdc).Result;


                dynamic resource = result.Content.ReadAsAsync<JObject>().Result;


                var resultDownload =
client.GetAsync((string)resource.PackagedApplicationDownloadUrl + "?subscription-key=" +
"033f092b-50ec-4f27-a157-2103fac22894").Result;

                File.WriteAllText(@"C:\TEMP\jsoutput.txt",
resultDownload.Content.ReadAsStringAsync().Result);

                return resultDownload.Content.ReadAsStringAsync().Result;
            }
        }
```

Step 4

The JavaScript that is returned can either be stored in memory or persisted to the device.

For More Information

For more information on how to integrate the JavaScript rule engine into an application, please see *InRule Integration Brief: Client-side Rule Execution*.

For more information on how to include InRule *for JavaScript* in a build process, please see *InRule Deployment Brief: Deploying JavaScript Rule Changes*.

For additional information please visit inrule.com, contact us or reach out to your InRule Account Executive to get in touch with a pre-sales engineer.